|  |
| --- |
| 哈尔滨工业大学(深圳) |
| **《数据结构与算法》实验报告** |
|  |
| 实验二  排序算法及树型结构  学 院: 计算机科学与技术学院   |  |  | | --- | --- | | 姓 名: | 赵辰瑞 | | 学 号: | 2023111656 | | 日 期: | 2024-11-14 | |

# 一、问题分析

将原题要解决的问题转换成用计算机要解决的问题。

染色问题：每次操作遍历树，最后寻找满足条件的节点，输出个数。

排序问题：用能将原始数据和映射数据相关联的存储方式，以一种稳定排序的方式对映射数据进行排序，就能打印所求的原始数据。

# 二、详细设计

## 2.1 设计思想

用自然语言描述解决问题的方案 。

1 尽管这是一棵二叉搜索树，但我们想要求的是树节点满足条件的个数，所以完全没必要使用树来存储。我将使用最为普通的数组来存储：

    int N, n, x, y, mode, count = 0;

    scanf("%d", &N);

    int \*nodes = calloc(N, sizeof(int));

    for (int i = 0; i < N; i++)

    {

        char input[10];

        scanf("%s", input);

        nodes[i] = (int)strtol(input, NULL, 10);

}

存储好数据后，发现：如果按照默认遍历顺序，最大遍历次数可达，这是难以接受的。决定优化方案。优化思路：从个数上进行优化。每次操作相当于覆盖，也就是说，对于任意一个节点，只有最后一次操作才有效。

于是想到这样一种方案：倒序检查操作，从后往前进行操作，每把符合条件的节点识别完（识别包含：是否处于内，以及是否为），就把它删除（删除即不加入到下一轮）。当本轮检查完，于是返回新数组。这些元素是不符合条件的：

for (int i = n - 1; i > -1; i--)

        nodes = filter\_array(nodes, N, &N, edges[i][1], edges[i][2], edges[i][0]);

其中filter\_array是根据需求定义的动态数组：

int\* filter\_array(const int \*array, const int size, int \*new\_size, const int x, const int y, const int mode)

{

    int \*new\_array = calloc(size, sizeof(int));

    if (!new\_array)

        return NULL;

    int count = 0;

    for (int i = 0; i < size; i++)

    {

        if (array[i] >= x && array[i] <= y)

                RedDeadRedemption += mode ? 0 : 1;

        else

            new\_array[count++] = array[i];

    }

    \*new\_size = count;

    return realloc(new\_array, count \* sizeof(int));

}

本函数没有释放new\_array，可能会导致内存泄漏。

2 核心部分就是求映射数值。对于每一个数位的操作，可以用字符串。其他算法：查阅资料发现一个除以10求余数的高级算法。字符串操作：

char str[12], mapped\_str[12];

    sprintf(str, "%d", num);

    int i = 0;

    while (str[i] != '\0')

        mapped\_str[i++] = (char)(mapping[str[i] - '0'] + (int)'0');

    mapped\_str[i] = '\0';

    return strtol(mapped\_str, NULL, 10);

先用sprintf把int换成char\*，再轮询每个数位，用映射规则把它们映射过去。strtol自动把前导0去掉了。

下一步排序：必须使用稳定排序，否则不清楚相等元素位置。我使用了归并排序。

## 2.2 存储结构及操作

(1) 存储结构：1 简单的数组，使用calloc；2 自定义结构体，这样可以把original和mapping绑定。也可以用二维数组，效果和占用内存是一样的，显示上没有结构体易懂。

(2) 涉及的操作：请看2.1设计思想。

## 2.3 程序整体流程图

手机屏幕截图

描述已自动生成文本

描述已自动生成

# 三、用户使用手册

1 输入

- Line 1: 第一行是“二叉树”数据个数，

- Line 2: 第二行是节点数值，-1代表这里是空

- Line 3: 一共进行的操作次数n

- Line 4: 以[]包围的、[int int int] [int,int,int]随机出现n次的字符串

输出：红色的节点个数

2 输入

- Line 1: 第一行是原始数据个数

- Line 2: 第二行是10个0~9的数，表示mapping

- Line 3: 以空格隔开的原始数据

输出：排序好的原始数据，以空格隔开，行末没有空格

# 四、结果

手机屏幕的截图

描述已自动生成文本

中度可信度描述已自动生成

# 五、总结

以我的编程思路看第一题，还是含有很多坑点的。首先“二叉树”一定是最大限制程序效率的坑点。我们进行的是统计操作，每一个节点到底在什么位置、和其他节点的关系完全没有意义，我们只需要考虑这个节点的“值”是什么，所以一个简单的顺序表就能解决该问题。

其次，OJ给出的测试中含有一个奇怪的格式：[int,int,int]即把空格换成了英文逗号。这一点使得在处理x, y, mode时需要多加一个“或”：

if (

sscanf(ptr, "[%d %d %d]", &mode, &x, &y) == 3 ||

sscanf(ptr, "[%d,%d,%d]", &mode, &x, &y) == 3)

再次，“操作”遍历的复杂性使得我们不得不考虑时间成本。针对每次操作的“下一次操作”有可能是覆盖的，且覆盖之后所有之前的操作均无效，我们可以断定：倒序查看x, y, mode并执行操作是最高效的。

完成了倒序，我们还能继续优化。我们知道了“覆盖”，那么我们就可以直接把操作过并且符合条件的元素（位于区间内，无论是涂蓝还是涂红）直接删掉，因为前序操作无效。

最后，别忘了释放指针，泄漏不是一件小事。

以这个思路回看第二题，相对来说就十分地简单。首先考虑的就是原始数据和映射数据应该用什么存储。它们必须绑定到一起，又便于访问，我第一个想到的是结构体（类）。至于为什么没有想到二维数组，我想是因为最近在练习栈、队列和二叉树，使用结构体(node)比较多。

使用结构体和使用二维数组在本质上没有区别，我们来做一下比较：

|  |  |  |  |
| --- | --- | --- | --- |
|  | struct | int[1][2] | details |
| pointer | 1 ptr | 1 ptr | access to that |
| data | 2 int | 2 int | original, mapped |
| data\_storage | 8 bytes | 8 bytes | (-> 2 of int) |

虽然访问方式有所区别，但它们占用空间是相同的。

考虑完存储，接下来应该考虑映射该如何实现。做题前查阅了力扣，发现了一个使用除以十求余数的计算办法，我认为是非常好的。但我没有想到，我使用字符串，即将数字先转换成字符串再一个一个访问。我想到字符串的原因很简单，因为我最近在自己实现 C++ 关于 string的一些功能。

接下来想到排序，题中又说需要保持，那么一定是稳定排序。刚上完《数据结构与算法》课，想到的第一个就是归并排序(merge\_sort)，因为刚学完分治，有这个思想。

所有核心均已实现完毕，最后别忘了看看句末有没有空格，并释放指针。